Registration No.

22646

-Technical Report-

GNU C/C++ AND FORTRAN LANGUAGE INTEROPERABILITY WITH FUNCTION USAGE

*****

Disclaimer: Reference herein to any specific commercial company, product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the Department of the Army (DoA). The opinions of the authors expressed herein do not necessarily state or reflect those of the United States Government or the DoA, and shall not be used for advertising or product endorsement purposes.

*****

UNCLASSIFIED: Distribution Statement A.

Approved for public release; distribution is unlimited.

March 2012

U.S. Army Tank Automotive Research,
Development, and Engineering Center
Detroit Arsenal
Warren, Michigan 48397-5000

TARDEC Technical Report No. 22646
March 2012

GNU C/C++ AND FORTRAN LANGUAGE INTEROPERABILITY WITH FUNCTION USAGE

W. Bylsma
Dynamics and Structures
U.S. Army Research, Development and Engineering Command (RDECOM)
U.S. Army Tank-automotive and Armaments Research, Development and Engineering Center (TARDEC)
Detroit Arsenal
ATTN: RDTA-RS/MS157
6501 East 11 Mile Road
Warren, Michigan 48397-5000

* * * * *

* * * * *

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	**2. REPORT TYPE**	**3. DATES COVERED** *(From - To)*
12-03-2012	TECHNICAL	2012

GNU C/C++ AND FORTRAN LANGUAGE INTEROPERABILITY WITH FUNCTION USAGE

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
WESLEY BYLSMA

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
DYNAMICS AND STRUCTURES-US ARMY RDECOM/TARDEC
ATTN: RDTA-RS/MS157
6501 E 11 MILE RD
WARREN, MI 48397-5000

8. PERFORMING ORGANIZATION REPORT NUMBER

22646

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT
Distribution Statement A: Approved for public release; distribution is unlimited.

13. SUPPLEMENTARY NOTES

14. ABSTRACT

This report documents GNU GCC options for interoperability between C/C++ and FORTRAN. Compiling legacy software on the Linux platform necessitated the usage of GNU compilers and changes to several intrinsic functions. Several miscellaneous related activities are also included. Experimentation was required to achieve success. Included are examples demonstrating particular usage of each. The functions addressed are TIME(), CTIME(), ETIME(), ITIME(), IDATE() (FORTRAN), getenv(), strncpy() (C), and vector<> (C++).

15. SUBJECT TERMS
GNU GCC, C/C++, FORTRAN, language interoperability

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**	Unclassified	18	Wesley Bylsma
Unclassified	Unclassified	Unclassified			**19b. TELEPHONE NUMBER** *(include area code)* 586-282-4104

Contents (by Title)

GNU C/C++ AND FORTRAN LANGUAGE INTEROPERABILITY WITH FUNCTION USAGE

TARDEC Technical Report No. 22646
March 2012

1.0 INTRODUCTION

This report documents GNU GCC options for interoperability between C/C++ and FORTRAN. Compiling legacy software on the Linux platform necessitated the usage of GNU compilers and changes to several intrinsic functions. Several miscellaneous related activities are also included. Experimentation was required to achieve success. Included are examples demonstrating particular usage of each. The functions addressed are TIME(), CTIME(), ETIME(), ITIME(), IDATE() (FORTRAN), getenv(), strncpy() (C), and vector<> (C++).

All examples were run on SUSE Linux Enterprise Server 11 (x86_64) with

- LSB_VERSION="core-2.0-noarch:core-3.2-noarch:core-4.0-noarch:core-2.0-x86_64:core-3.2-x86_64:core-4.0-x86_64"
- SGI ProPack 7SP1 for Linux, Build 701r3.sles11-1005252113
- SGI Tempo Service Node 2.1, Build 701r3.sles11-1005252113
- VERSION = 11
- PATCHLEVEL = 1

and using the GNU utilities

- gcc (SUSE Linux) 4.3.4 [gcc-4_3-branch revision 152973]
- GNU Fortran (SUSE Linux) 4.3.4 [gcc-4_3-branch revision 152973]
- GNU ld (GNU Binutils; SUSE Linux Enterprise 11) 2.20.0.20100122-0.7.9
- GNU ar (GNU Binutils; SUSE Linux Enterprise 11) 2.20.0.20100122-0.7.9

See references [1], [2], and [3].

2.0 C/C++ CALLING FORTRAN

This section demonstrates how to call FORTRAN subroutines from C/C++. A good reference is [4] that highlights these necessary considerations:

- extern "C" directive is used for modules not written in C++
- Fortran symbol references must be lowercase in C++ calls (C++ is case sensitive, Fortran is not)
- Usually Fortran appends an underscore (_) at the end of definitions and references to external symbols (subroutines, functions, etc.)
- C++ passes all parameters by value (except arrays and structures). Fortran passes them by reference

2.1 CHECK FOR 32 BIT AND 64 BIT LIBRARIES

This example shows how to call FORTRAN from C and tests for compatibility between 32 bit and 64 bit objects.

```
echo "\n\n"
echo "1-both 32-bit"
rm test.1
gfortran -m32 -c u.f
gcc -m32 u.o u.c -o test.1 -lgfortran
test.1

echo "\n\n"
echo "2-.f 32-bit"
rm test.2
gfortran -m32 -c u.f
gcc u.o u.c -o test.2 -lgfortran
test.2

echo "\n\n"
echo "3-both 64 bit"
rm test.3
gfortran -c u.f
gcc u.o u.c -o test.3 -lgfortran
test.3
```

u.c

```
#include <stdio.h>
extern test_();
int main() {
   printf("u.c: before call to test_\n");
   test_();
   printf("u.c: after call to test_\n");
}
```

u.f

```
  SUBROUTINE test
      print *,'in u.f'
  END
```

OUTPUT:

```
1-both 32-bit
u.c: before call to test_
 in u.f
u.c: after call to test_


2-.f 32-bit
rm: cannot remove `test.2': No such file or directory
/usr/lib64/gcc/x86_64-suse-linux/4.3/../../../../x86_64-suse-linux/bin/ld:  i386  architecture
of input file `u.o' is incompatible with i386:x86-64 output
collect2: ld returned 1 exit status
test.2: Command not found.


3-both 64 bit
u.c: before call to test_
 in u.f
u.c: after call to test_
```

2.2 FORTRAN TIME() AND CTIME() USAGE

This example shows how to call FORTRAN from C and the usage of the FORTRAN TIME() and CTIME() intrinsic functions (see [5] and [6]). The code "u.c" in section 2.1 is used.

```
rm test.1
gfortran -c ut.f
gcc ut.o ../section2.1/u.c -o test.1 -lgfortran
test.1
```

ut.f

```
   SUBROUTINE test
   CHARACTER*30 TimeSTR
   CHARACTER*8 TimeSTR1
   CALL CTIME(TIME(),TimeSTR)
   print *,'TimeSTR:[',TimeSTR,']'
   TimeSTR1 = TimeSTR(12:19)
   print *,'TimeSTR1:[',TimeSTR1,']'
   print *,'TIME():',TIME()
   END
```

OUTPUT:

```
u.c: before call to test_
 TimeSTR:[Wed Mar  7 09:56:38 2012      ]
 TimeSTR1:[09:56:38]
 TIME():  1331132198
u.c: after call to test_
```

2.3 COMBINING OBJECTS INTO ONE EXECUTABLE

This example shows how to call FORTRAN from C, compile each object separately, and then combine the objects into an executable. The code in section 2.2 is used.

```
gfortran -m32 -c ../section2.2/ut.f -o f1.o
gcc -m32 -c ../section2.2/u.c -o c1.o
gcc -m32 f1.o c1.o -o cftest.e -lgfortran
cftest.e
```

The output below should be same output as above except for the time.

OUTPUT:

```
u.c: before call to test_
 TimeSTR:[Wed Mar  7 10:04:12 2012      ]
 TimeSTR1:[10:04:12]
 TIME():  1331132652
u.c: after call to test_
```

2.4 C GETENV USAGE

This example shows the usage of the C GETENV() function (see [7]) and string manipulation using the tcsh shell to set the environment variable.

```
#!/usr/bin/tcsh
gcc uenv.c -o uenv.e
echo "\n\n===setenv"
setenv TESTENV /path1/path2/file
uenv.e
echo "\n\n===unsetenv"
unsetenv TESTENV
uenv.e
```

uenv.c

```
#include <stdio.h>
#include <stdlib.h> /* getenv */
#include <string.h> /* strcpy */
int main() {
  char *str = "DUMMYENV";
   char *str1;
   char str2[50];
   printf("main: begin\n");
   printf("str:%s\n",str);
   str1 = getenv("TESTENV");
   if (str1 == NULL)
   {
      printf("NULL\n");
   } else
   {
      printf("str1:%s\n",str1);
     printf("len1:%d\n",strlen(str1));
   }
   strcpy(str2,str);
   str2[strlen(str2)-2] = 'X';
   str = str2;
   printf("str:%s\n",str);
   printf("len:%d\n",strlen(str));
   printf("main: end\n");
}
```

OUTPUT:

```
===setenv
main: begin
str:DUMMYENV
str1:/path1/path2/file
len1:17
str:DUMMYEXV
len:8
main: end

===unsetenv
main: begin
str:DUMMYENV
NULL
str:DUMMYEXV
len:8
main: end
```

3.0 FORTRAN CALLING C/C++

This section demonstrates how to call C/C++ subroutines from FORTRAN. A good reference is [8] and [9] where consideration is given to C++ name mangling (a technique used to allow function overloading due to C++'s object orientated foundations). Also notice the appended underscore (_) as noted in section 2.0 for FORTRAN function calls.

3.1 FORTRAN ETIME() USAGE

This example shows how to call C from FORTRAN and the usage of the FORTRAN ETIME() intrinsic function (see [10]).

```
rm f1.o c1.o fctest.e
gfortran -m32 -c u1.f -o f1.o  -lgfortran
gcc -m32 -c u1.c -o c1.o
echo "=====nm"
echo "f1.o"
nm f1.o
echo "---"
echo "c1.o"
nm c1.o
echo "====="
gfortran -m32 f1.o c1.o -o fctest.e
fctest.e
```

u1.c

```
#include <stdio.h>
int pp_(int *a) {
   int k;
   printf("pp: begin\n");
   k = *a;
   printf("k:%d\n",k);
   printf("a*:%d\n",*a);
   printf("pp: end\n");
}
```

u1.f

```
  PROGRAM main
  REAL(4) result, TARRAY(2)
  INTEGER LIC
  print *,'main: begin'
  LIC = 1000000
  CALL ETIME(TARRAY,result)
  print *,'res:',result
  print *,'ta1:',tarray(1)
  print *,'ta2:',tarray(2)
  print *,' etime: start'
  print *,'LIC:',LIC
  CALL pp(LIC)
  ii = -55
  print *,'ABS(ii):',ABS(ii)
  do i=1,100*LIC
    j=i*i-i
  enddo
  print *,' etime: end'
  CALL ETIME(TARRAY,result)
  print *,'res:',result
  print *,'ta1:',tarray(1)
  print *,'ta2:',tarray(2)
  print *,'main: end'
  END
```

OUTPUT:

```
---call C++
=====nm
f1.o
00000000 T MAIN__
         U _gfortran_etime_sub
         U _gfortran_set_options
         U _gfortran_st_write
         U _gfortran_st_write_done
         U _gfortran_transfer_character
         U _gfortran_transfer_integer
         U _gfortran_transfer_real
00000000 r options.0.547
         U pp_
c1.o
00000000 T pp_
         U printf
         U puts
=====
 main: begin
 res:   0.0000000
 ta1:   0.0000000
 ta2:   0.0000000
  etime: start
 LIC:     1000000
pp: begin
k:1000000
a*:1000000
pp: end
 ABS(ii):          55
  etime: end
 res:  0.46802899
 ta1:  0.46802899
 ta2:   0.0000000
 main: end
```

This example shows how to call C++ from FORTRAN. The code “u1.f” above is used.

```
rm f1.o c1.o fcpptest.e
echo "---call C++"
gfortran -m32 -c ../section3.1/u1.f -o f1.o  -lgfortran
gcc -m32 -c u1.cpp -o c1.o  -lstdc++
echo "=====nm"
echo "f1.o"
nm f1.o
echo "---"
echo "c1.o"
nm c1.o
echo "====="
gfortran -m32 f1.o c1.o -o fcpptest.e  -lstdc++
fcpptest.e
```

u1.cpp

```
#include <iostream>
using namespace std;
extern "C" {
   int pp_(int *a) {
   int k;
   cout << "pp: begin" << endl;
   k = *a;
   cout << "k:" << k << endl;
   cout << "a*:" << *a << endl;
   cout << "pp: end" << endl;
   return 0;
   }
}
```

OUTPUT:

```
=====nm
f1.o
00000000 T MAIN__
         U _gfortran_etime_sub
         U _gfortran_set_options
         U _gfortran_st_write
         U _gfortran_st_write_done
         U _gfortran_transfer_character
         U _gfortran_transfer_integer
         U _gfortran_transfer_real
00000000 r options.0.547
         U pp_
---
c1.o
00000040 t _GLOBAL__I_pp_
00000000 t _Z41__static_initialization_and_destruction_0ii
         U _ZNSolsEPFRSoS_E
         U _ZNSolsEi
         U _ZNSt8ios_base4InitC1Ev
         U _ZNSt8ios_base4InitD1Ev
         U _ZSt4cout
         U _ZSt4endlIcSt11char_traitsIcEERSt13basic_ostreamIT_T0_ES6_
00000000 b _ZStL8__ioinit
         U _ZStlsISt11char_traitsIcEERSt13basic_ostreamIcT_ES5_PKc
         U __cxa_atexit
         U __dso_handle
         U __gxx_personality_v0
0000005c T pp_
=====
 main: begin
 res:   0.0000000
 ta1:   0.0000000
 ta2:   0.0000000
  etime: start
 LIC:     1000000
pp: begin
k:1000000
a*:1000000
pp: end
 ABS(ii):          55
  etime: end
 res:  0.45602801
 ta1:  0.45602801
 ta2:   0.0000000
 main: end
```

3.2 FORTRAN ITIME() AND IDATE USAGE

This example shows how to call C++ from FORTRAN and the usage of the FORTRAN ITIME() and IDATE() intrinsic functions (see [11] and [12]). The values returned in each array are summarized below:

ITIME	IDATE
1. hour, 1-24	1. day, 1-31
2. minute 1-60	2. month, 1-12
3. second 1-60	3. year

```
rm utcpp.o udcpp.o ut.o ut.e
gcc -m32 -c ut.cpp -o utcpp.o -lstdc++
gcc -m32 -c ud.cpp -o udcpp.o -lstdc++
gfortran -m32 -c ut.f -o ut.o -lgfortran
gfortran -m32 ut.o utcpp.o udcpp.o -o ut.e -lstdc++
ut.e
```

ut.cpp

```
#include <iostream>
using namespace std;
extern "C" {
   int pt_(int *a) {
      cout << "pt: begin" << endl;
      cout << "  hour:" << *a << endl;
      cout << "minute:" << *(a+1) << endl;
      cout << "second:" << *(a+2) << endl;
      cout << "pt: end" << endl;
      return 0;
   }
}
```

ud.cpp

```
#include <iostream>
using namespace std;
extern "C" {
   int pd_(int *a) {
      cout << "pd: begin" << endl;
      cout << "  day:" << *a << endl;
      cout << "month:" << *(a+1) << endl;
      cout << " year:" << *(a+2) << endl;
      cout << "pd: end" << endl;
      return 0;
   }
}
```

ut.f

```
   PROGRAM main
   INTEGER TARRAY(3)
   print *,'main: begin'
   CALL ITIME(TARRAY)
   CALL pt(tarray)
   print *,'ta1:',tarray(1)
   print *,'ta2:',tarray(2)
   print *,'ta3:',tarray(3)
   CALL IDATE(TARRAY)
   CALL pd(tarray)
   print *,'ta1:',tarray(1)
   print *,'ta2:',tarray(2)
   print *,'ta3:',tarray(3)
   print *,'main: end'
   END
```

OUTPUT:

```
main: begin
pt: begin
  hour:10
minute:59
second:20
pt: end
 ta1:          10
 ta2:          59
 ta3:          20
pd: begin
  day:12
month:3
 year:2012
pd: end
 ta1:          12
 ta2:           3
 ta3:        2012
 main: end
```

3.3 FORTRAN STRING MANIPULATION USING C

This example shows how to call C from FORTRAN and the usage of C string functions to manipulate FORTRAN strings.

```
rm f1.o c1.o fstr.e
gcc -m32 -c ustr.c -o c1.o
gfortran -m32 -c ustr.f -o f1.o  -lgfortran
gfortran -m32 f1.o c1.o -o fstr.e
fstr.e
```

ustr.c

```
#include <stdio.h>
#include <string.h> /*strlen*/
int pp_(char *vi, int *lvi, char* vo, int *lvo) {
   char tmpstr[250];
   char fmt[250];
   char tt[] = "/path1/path2/file";
   printf("pp: start\n");
   printf("lvi:%d\n",*lvi);
   printf("lvo:%d\n",*lvo);
   strncpy(tmpstr,vi,*lvi);
   tmpstr[*lvi] = '\0';
   printf("vi:[%s]\n",tmpstr);
   strncpy(vo,tt,strlen(tt));
   *lvo = strlen(tt);
   sprintf(fmt,"format:%%%dc\0",6);
   printf("fmt:[%s]\n",fmt);
   sprintf(tmpstr,fmt,' ' );
   printf("fmt:[%s]\n",tmpstr);
   printf("pp: end\n");
   return;
}
```

ustr.f

```
    PROGRAM main
    CHARACTER*250 TMPSTR
    INTEGER varlen,vallen
    print *,'main: begin'
    varlen = 5
    vallen = 5
    write(TMPSTR,'(A250)')' '
    TMPSTR = '/home1/home2/home3/file4'
    print *,'TMPSTR:[',TMPSTR(:vallen),']'
    print *,'TMPSTR LEN:',vallen,LEN(TMPSTR)
    CALL pp('CADSI',varlen,TMPSTR,vallen)
    print *,'TMPSTR:[',TMPSTR(:vallen),']'
    print *,'TMPSTR LEN:',vallen , LEN(TMPSTR)
    print *,'main: end'
    END
```

OUTPUT:

```
main: begin
 TMPSTR:[/home]
 TMPSTR LEN:           5         250
pp: start
lvi:5
lvo:5
vi:[CADSI]
fmt:[format:%6c]
fmt:[format:      ]
pp: end
 TMPSTR:[/path1/path2/file]
 TMPSTR LEN:          17         250
 main: end
```

3.4 FORTRAN ARRAY MANIPULATION USING C++

This example shows how to call C++ from FORTRAN and the usage of the C++ vector<> standard template library (STL) to manipulate FORTRAN arrays (see [13]).

```
gfortran -m32 -c vf.f -o vf.o  -lgfortran
gcc -m32 -c vc.cpp -o vc.o -lstdc++
gfortran -m32 vf.o vc.o -o vfcpptest.e -lstdc++
vfcpptest.e
```

vc.cpp

```
#include <iostream>
#include <vector>
using namespace std;
extern "C" {
   int pa_(int *siz, double *arr) {
      int dim,sum  = 1;
      vector<double> a;
      printf("pa: begin\n");
      dim = *(siz);
      cout << "dim:" << dim << endl;
      for (int i=1;i<=dim;i++) {
         cout << i << " " <<  *(siz+i) << endl;
         sum = sum * (*(siz+i));
      }
      cout << "sum:" << sum << endl;
      for (int i=1;i<=sum;i++) {
         a.push_back(i);
      }
      cout << "a.size:" << a.size() << endl;
      for(int i=0;i<sum;i++) {
         *(arr+i) = a[i];
      }
      printf("pa: end\n");
   }
}
```

vf.f

```
   PROGRAM main
   DOUBLE PRECISION arr(3,2),arr1(3,2,3)
   INTEGER i,j,siz(4)
   print *,'main: begin'
   print *,'arr(5,2) = arr(i,j)'
   print *,'     j=1,2'
   arr(3,2) = -99.6789
   siz = (/2, 3, 2, 0/)
   do i=1,siz(2)
      print *, 'i=',i,' ',arr(i,1),' ',arr(i,2)
   enddo
   CALL pa(siz,arr)
   do i=1,siz(2)
      print *,'i=',i,' ',arr(i,1),' ',arr(i,2)
   enddo
   print *,''
   print *,'arr1(3,2,3) = arr1(i,j,k)'
   print *, 'i,j,k, val'
   arr1(3,2,3) = -99.6789
   siz = (/3, 3, 2, 3/)
   do k=1,siz(4)
      do j=1,siz(3)
         do i=1,siz(2)
            print *, i,j,k,arr1(i,j,k)
         enddo
      enddo
   enddo
   CALL pa(siz,arr1)
   do k=1,siz(4)
      do j=1,siz(3)
         do i=1,siz(2)
            print *, i,j,k,arr1(i,j,k)
         enddo
      enddo
   enddo
   print *,'main: end'
   END
```

OUTPUT:

```
main: begin
 arr(5,2) = arr(i,j)
      j=1,2
 i=           1   -1.92681655437397497E+220    6.64588727654947730E-316
 i=           2    4.090883488054987 54E-314    2.18846198183111329E-314
 i=           3    4.94065645841246544E-324    -99.678901672363281
pa: begin
dim:2
1 3
2 2
sum:6
a.size:6
pa: end
 i=           1    1.00000000000000000          4.0000000000000000
 i=           2     2.0000000000000000          5.0000000000000000
 i=           3     3.0000000000000000          6.0000000000000000

 arr1(3,2,3) = arr1(i,j,k)
 i,j,k, val
           1           1           1   0.0000000000000000
           2           1           1 -1.92634262907847196E+220
           3           1           1  6.64588727654947730E-316
           1           2           1   0.0000000000000000
           2           2           1   0.0000000000000000
           3           2           1   0.0000000000000000
           1           1           2   0.0000000000000000
           2           1           2   0.0000000000000000
           3           1           2   0.0000000000000000
           1           2           2   0.0000000000000000
           2           2           2   0.0000000000000000
           3           2           2   0.0000000000000000
           1           1           3   0.0000000000000000
           2           1           3   0.0000000000000000
           3           1           3 -1.78035013165935472E+220
           1           2           3                    NaN
           2           2           3 -1.92634255302512759E+220
           3           2           3  -99.678901672363281
pa: begin
dim:3
1 3
2 2
3 3
sum:18
a.size:18
pa: end
           1           1           1  1.00000000000000000
           2           1           1   2.0000000000000000
           3           1           1   3.0000000000000000
           1           2           1   4.0000000000000000
           2           2           1   5.0000000000000000
           3           2           1   6.0000000000000000
           1           1           2   7.0000000000000000
           2           1           2   8.0000000000000000
           3           1           2   9.0000000000000000
           1           2           2  10.0000000000000000
           2           2           2   11.000000000000000
           3           2           2   12.000000000000000
           1           1           3   13.000000000000000
           2           1           3   14.000000000000000
           3           1           3   15.000000000000000
           1           2           3   16.000000000000000
           2           2           3   17.000000000000000
           3           2           3   18.000000000000000
 main: end
```

4.0 MISCELLANEOUS

This section demonstrates several miscellaneous items associated with compiling C/C++ and FORTRAN.

4.1 PREPROCESSOR INCLUDE STATEMENTS

This example shows how to use include header files with FORTRAN.

```
gfortran -xf77-cpp-input ftest.f -o ftest
ftest
```

itest.h

```
    INTEGER a
    print *,'itest.h: set a to 15'
    a = 12
    a = 15
```

ftest.f

```
    PROGRAM test
    #include "itest.h"
    print *,'test: before a'
    print *,'a=',a
    print *,'test: after a'
    END
```

OUTPUT:

```
  itest.h: set a to 15
  test: before a
  a=          15
  test: after a
```

4.2 LARGE FILE SUPPORT (LSF)

This example shows how to include Large File Support (LSF) to read files greater than 2 GB on 32-bit systems (see [14], [15], and [16]). Programs have to be recompiled with “-D_FILE_OFFSET_BITS=64" which forces “off_t” and file access to 64 bits. An example with casting is below.

```
(off_t) curpos = lseek(fd, (off_t)0, SEEK_CUR);
```

```
clear
echo "===32-bit"
gcc -m32 offsize.c -o csize
gcc -m32 offsize.cpp -o cppsize -lstdc++
csize
cppsize
echo ""
echo "===32-bit, w/-D_FILE_OFFSET_BITS=64"
gcc -m32 offsize.c -o cwsize -D_FILE_OFFSET_BITS=64
gcc -m32 offsize.cpp -o cppwsize -lstdc++  -D_FILE_OFFSET_BITS=64
cwsize
cppwsize
```

```
rm csize cppsize cwsize cppwsize
echo ""
echo "===64-bit"
gcc offsize.c -o csize
gcc offsize.cpp -o cppsize -lstdc++
csize
cppsize
```

offsize.c

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc, const char* argv[]) {
   printf("in main C: begin\n");
   printf("     sizeof(char):%d\n",sizeof(char));
   printf("sizeof(short int):%d\n",sizeof(short int));
   printf("      sizeof(int):%d\n",sizeof(int));
   printf("     sizeof(long):%d\n",sizeof(long));
   printf("    sizeof(off_t):%d\n", sizeof(off_t));
   printf("in main C: end\n");
}
```

offsize.cpp

```
#include <iostream>
#include <ios>
using namespace std;
int main(int argc, const char* argv[]) {
   cout << "in main C++: begin" << endl;
   cout << "     sizeof(char):" << sizeof(char) << endl;
   cout << "sizeof(short int):" << sizeof(short int) << endl;
   cout << "      sizeof(int):" << sizeof(int) << endl;
   cout << "     sizeof(long):" << sizeof(long) << endl;
   cout << " sizeof(off_t) is:" << sizeof(off_t) << endl;
   cout << "sizeof(ios::off_type):" << sizeof(ios::off_type) << endl;
   return 0;
}
```

The output below shows the difference in size of "`off_t`" with the "`-D_FILE_OFFSET_BITS=64`" flag included---the C++ type "`ios::off_type`" is always 8 bytes in size.

OUTPUT:

```
===32-bit
in main C: begin
     sizeof(char):1
sizeof(short int):2
      sizeof(int):4
     sizeof(long):4
    sizeof(off_t):4
in main C: end
in main C++: begin
     sizeof(char):1
sizeof(short int):2
      sizeof(int):4
     sizeof(long):4
 sizeof(off_t) is:4
sizeof(ios::off_type):8

===32-bit, w/-D_FILE_OFFSET_BITS=64
in main C: begin
     sizeof(char):1
sizeof(short int):2
      sizeof(int):4
     sizeof(long):4
    sizeof(off_t):8
in main C: end
in main C++: begin
     sizeof(char):1
sizeof(short int):2
      sizeof(int):4
     sizeof(long):4
 sizeof(off_t) is:8
sizeof(ios::off_type):8

===64-bit
in main C: begin
     sizeof(char):1
sizeof(short int):2
      sizeof(int):4
     sizeof(long):8
    sizeof(off_t):8
in main C: end
in main C++: begin
     sizeof(char):1
sizeof(short int):2
      sizeof(int):4
     sizeof(long):8
 sizeof(off_t) is:8
sizeof(ios::off_type):8
service0 557%
```

4.3 CREATING AND COMPILING WITH ARCHIVES

This example shows how to create an archive and include it when compiling in the tcsh shell.

```
#!/usr/bin/tcsh
gcc -m32 -c s.c s1.c s2.c
echo "Creating Archive:"
ar rlusv libtest.a s1.o s2.o
true libtest.a
echo " "
echo "Listing Archive:"
ar -tv libtest.a
nm libtest.a
echo " "
gcc -m32 s.o libtest.a -o ss.e
#gcc -m32 s.o -o ss.e -L. -liit.a
ss.e
```

s1.c

```
#include <stdio.h>
void s1() {
   printf("in s1\n");
}
```

s2.c

```
#include <stdio.h>
void s2() {
   printf("in s2\n");
}
```

s.c

```
#include <stdio.h>
extern void s1();
extern void s2();
int main() {
   printf("Hello \n");
   s1();
   s2();
   printf("Hello \n");
}
```

OUTPUT:

```
Creating Archive:
ar: creating libtest.a
a - s1.o
a - s2.o

Listing Archive:
rw-r----- 2048/14900    932 Mar 12 13:38 2012 s1.o
rw-r----- 2048/14900    932 Mar 12 13:38 2012 s2.o

s1.o:
         U puts
00000000 T s1

s2.o:
         U puts
00000000 T s2

s: before s1,s2
in s1
in s2
s: after s1,s2
```

REFERENCES

[1] http://www.suse.com

[2] http://www.gnu.org

[3] http://gcc.gnu.org/

[4] http://wwwcompass.cern.ch/compass/software/offline/software/fandc/fandc.html

[5] http://gcc.gnu.org/onlinedocs/gfortran/TIME.html

[6] http://gcc.gnu.org/onlinedocs/gfortran/CTIME.html

[7] http://www.cplusplus.com/reference/clibrary/cstdlib/getenv/

[8] http://www.yolinux.com/TUTORIALS/LinuxTutorialMixingFortranAndC.html

[9] http://www.yolinux.com/TUTORIALS/LinuxTutorialC++.html#CANDCPP

[10] http://gcc.gnu.org/onlinedocs/gfortran/ETIME.html

[11] http://gcc.gnu.org/onlinedocs/gfortran/ITIME.html

[12] http://gcc.gnu.org/onlinedocs/gfortran/IDATE.html

[13] http://www.cplusplus.com/reference/stl/vector/

[14] http://www.suse.de/~aj/linux_lfs.html

[15] http://stackoverflow.com/questions/965725/large-file-support-in-c

[16] http://groups.google.com/group/comp.lang.c++.moderated/browse_thread/thread/f4be9c112c05ef9b